Nonlinear Analysis: Modelling and Control, Vilnius, IMI, 1998, No 3

SYMBOLIC APPROXIMATION FOR COMPUTER VISION

Richard E. Blake
Norwegian University of Science and Technology, N-7034, Trondheim, Norway

Abstract

The need for a definition of discrete convergence in matching problems, which are essential in computer vision, is described and the fundamental properties of Scott lattice theory are outlined. Three data types: relational graphs, graph match_table and constraints, are considered and partial orderings are exhibited for them. A matching iteration consistent with the theory is sketched.

INTRODUCTION

The task of the matching activity in computer vision is to label fragments of the image data by associating them with a reference. The matching of features such as relational graphs is a discrete process that builds lists of acceptable labelings, [1]. The matching will generally be an NP problem and will not be tractable without a control strategy that exploits constraints and problem partitions, [2].

This paper proposes that matching can be viewed as a converging discrete iteration which terminates at (or near) a fixed point. A clear concept of convergence must first be established and it should then be possible to monitor progress as one aspect of a control strategy that essentially allocates computing resources.

The general expectation in executing a computer program is that, as computing effort is expended, there should be visible progress towards a result. This view is most familiar in considering numerical algorithms where progress to convergence can be studied analytically. It could be said that numerical algorithms are only 'respectable' if convergence can be proven.

Non-numerical algorithms also expend computing effort and should deliver a result in a non-numeric data type. Issues such as the quality of the final result and the improving quality of intermediate results are much less discussed than for numerical algorithms. The question of convergence for non-numerical algorithms is very general. This author conjectures that the reason the topic is neither better developed, nor applied in practice, is that each application brings with it a large consideration of fine detail of the problem domain. The present paper seeks to investigate graph matching from this point of view.

Lattice theory is put forward as the theoretical base. This was originally developed

by Scott and others for denotational semantics, see for example [3]. The relevant parts of the theory are introduced in the next section.

1. INTRODUCTION TO SCOTT LATTICE THEORY

The theory requires that computable functions and the data types that they involve have two key features:

1. that it must be possible to define a partial ordering for the data types in use, and,
2. that each computable function must be order preserving, that is a higher input point from the lattice that is its domain cannot lead to a lower output point in the lattice that is its range.

Formally, given $f: D \rightarrow D'$ and $x_1, x_2 \in D$, then $x_1 \subseteq_D x_2$ implies $f(x_1) \subseteq_{D'} f(x_2)$ where $\subseteq_D$ and $\subseteq_{D'}$ are the partial ordering relations in D and D'.

A desirable property is that the function should gain; that is, given $f: D \rightarrow D$ and $x_1, x_2, \ldots x_p \in D$, and with $x_2 = f(x_1); x_3 = f(x_2); x_4 = f(x_3) \ldots$, the x values are ordered: $x_1 \subseteq_D x_2 \subseteq_D x_3 \subseteq_D \ldots$

With care, order preservation and gaining can be defined for functions of more than one argument.

The overall top point is denoted by $\top$ and the overall lowest point is denoted by $\bot$, [3]. The highest x value below $\top$ in the lattice is the most precise result, [3]. The position of a working point in the lattice shows how good an approximation that point is to the most precise result. As the iteration runs, order preservation implies that the working point cannot fall. 'Convergence' has occurred if the iteration stabilises at a 'high enough' working point.

A least upper bound operation, ***LUB***, will exist. The ***LUB*** is essentially a structural merging of the information contained in the points it is applied to. The result is an improved point that combines the information in the input points.

Properties of the ***LUB*** include:

with $x_1, x_2, x_3 \in D$,

1. if $x_3 = \text{LUB}(x_1, x_2)$ then $x_1 \subseteq_D x_3$ and $x_2 \subseteq_D x_3$;
2. if $x_1 \subseteq_D x_2$ then $x_2 = LUB(x_1, x_2)$;
3. the ***LUB*** operation returns $\top$ if the two lists were not compatible, [3].

2. THE DATA TYPES FOR A MATCHING ITERATION AND THE 'MATCH' MAPPING

Three data types are needed for the matching operation: graph, match_table and

constraint. They have the uses:

- graph: domain of graphs to be matched;
- match_table: domain of tables indicating (sub)graph associations;
- constraint: domain of constraints that define partitions of the problem.

The first step in applying the theory is to examine the domain and range of the match activity. It is responsible for forming an association between two graphs, g_x and g_y, under the partitioning defined by the constraint, c. The match will typically minimise the cost of association, [7]. It is also assumed that the matching is incremental: the results of previous iterations are contained in the initial match_table, m_1, so that the solution can be built up in steps, [5]. The result is an improved match_table, m_2, this is represented by: m_2 = match (m_1, g_x, g_y, c), and match is the mapping: *match_table ´graph ´graph ´constraint ®match_table*.

The match activity is a partial function and cannot be applied to arbitrary combinations of arguments. Its partialness is protected by applying it in the correct sequence with other activities. The iteration creates a trajectory in the space:
match_table ´graph ´constraint.

The operations discussed below are only valid for points on possible trajectories.

Partial orderings will now be established for the data types by defining a comparison between two elements of each type and then showing that the relation is reflexive, transitive and anti-symmetric. A lemma, useful for showing the partial orderings, will first be proved.

Lemma 1: *Given that* $_ia \hat{I} D_i$ *is a set partially ordered by* $\subseteq_{D_i}$ *then another set,* D, *containing* D_I *elements,* x, *that are* n*-tuples of elements from the* D_i, *is partially ordered by* $\subseteq_D$, *with* $x_1 = <{}_1a_1, {}_2a_1, \ldots {}_ia_1, \ldots {}_{k1}a_1>$ *and* $x_2 = <{}_1a_2, {}_2a_2, \ldots {}_ia_2, \ldots {}_{k2}a_2>$
then $x_1 \subseteq_D x_2$ *iff* $k_1 £ k_2$ *and for all* $i, 1£i£k_1, {}_ia_1 \subseteq_D {}_ia_2$

Proof: Transivity, reflexivity and anti-symmetry will be established for $\subseteq_D$.
Transitivity: if $x_1 \subseteq_D x_2$ then $k_1 £ k_2$ and for all $i, 1£i£k_1, {}_ia_1 \subseteq_D {}_ia_2$; if $x_2 \subseteq_D x_3$ then $k_2 £ k_3$ and for all $i, 1£i£k_2, {}_ia_2 \subseteq_D {}_ia_3$. It follows immediately that $k_1 £ k_3$ and for all $i, 1£i£k_1, {}_ia_1 \subseteq_D {}_ia_2 \subseteq_D {}_ia_3$ so $x_1 \subseteq_D x_3$ establishing transitivity of $\subseteq_D$.
Reflexivity: Consider $x_1 \subseteq_D x_2$ so that $k_1 £ k_2$ and for all $i, 1£i£k_1, {}_ia_1 \subseteq_D {}_ia_2$ and the case $k_1 = k_2$ and ${}_ia_1 = {}_ia_2$ for all $i, 1£i£k_1$, it follows that $x_1 = x_2$ so that $\subseteq_D$ is reflexive.
Anti-symmetry: If $x_1 \subseteq_D x_2$ then $k_1 £ k_2$, for all $i, 1£i£k_1, {}_ia_1 \subseteq_D {}_ia_2$. If $x_2 \subseteq_D x_1$ then $k_2 £ k_1$ and for all $i, 1£i£k_2, {}_ia_2 \subseteq_D {}_ia_1$. It follows that $k_1 = k_2$ and for all $i, 1£i£k_1, {}_ia_1 = {}_ia_2$ from the reflexivity of $\subseteq_D$. Thus $x_1 = x_2$ establishing anti-symmetry of $\subseteq_D$. Since $\subseteq_D$ is transitive, reflexive and anti-symmetric it is a partial ordering relation. **q.e.d.**

Corollary: *If* $y = < \ldots {}_ix \ldots >$ *and* ${}_ix = < \ldots {}_i(a) \ldots >$ *then the existence of partial*

orderings for each of the $_{i}(_{j}a)$ *implies the partial ordering of the* y *values.*

The result is obtained by repeatedly applying the lemma to n-tuple components at successively deeper levels.

2.1. THE GRAPH TYPE AND ITS PARTIAL ORDERING

Graphs will be introduced first. A graph is defined as a collection of nodes, N, and a set of arcs, A. A graph, g, is $g = < N, A >$. Graphs are frequently augmented by attaching attributes to the nodes and arcs to increase their descriptive power for patterns. The attributes can be thought of as footnotes that give additional information about the interpretation of the node or arc in the pattern.

The node attributes can be paired with the node name to completely describe each node in N. Thus an individual node, $n, n \hat{I} N$, is n = <n_name, attribute_of_n_name>. and N is a set of pairs: $N = \{ ..., <$n_name, attribute_of_n_name$>, ...\}$. The attribute_of_n_name is a list of attribute-value pairs, [4], as will be illustrated below.

Arcs join a pair of nodes so that the pair, $<n_1,n_2> \hat{I} N \acute{} N$, is a natural part of the arc description. Attribute strings can also be attached to the arcs. A single arc can be expressed as: $a = < <n_1, n_2>$, attribute_of_arc_n_1_n_2 >. The set of arcs is then $A= \{ .. < <n_1, n_2>$, attribute_of_arc_n_1_n_2> $... \}$ The attribute_of_arc_n_1_n_2 is also a list of attribute-value pairs.

It is convenient to define selectors, $First, Second, \ldots, N^{th}$, which, applied to an n-tuple, return the appropriate element. It is also helpful to adopt the convention that selectors return a single element when applied to a single n-tuple, but when applied to a set of n-tuples, return a set of elements. This is a slight abuse of notation but does not lead to any serious errors. For example, $First(<a,b,c>) = a$, and $First(\{ \ldots <a,b,c>... \}) = \{ ... a, \ldots \}$.

A first step in defining a partial ordering for graphs is given in [6]. In outline, a grammar is defined that represents the graph by deriving all possible walks over it. The partial ordering is established for the grammars and thereby for the graphs. However, the graphs discussed in [6] are without attributes. The definition of the partial ordering can be extended to include the attributes. It is assumed that the partial ordering, $\subseteq_{Ug}$ from [6], exists for non-attributed relational graphs. The first step in the extension is to define a comparison $\subseteq_{At}$ for attribute strings. Both node and arc attributes can be considered at the same time: they are lists of propositions with parameters. The attribute string can be expressed in the form:

$$attribute\_string = \{ \ldots < proposition, <p_1, <p_2,...>>> \ldots \}$$

Consider the relation between two strings, s_1 and s_2, that pass both the following

tests:

1. s_2 contains all the propositions that s_1 contains,
2. for each proposition that s_1 contains with a parameter, the corresponding parameter for s_2 is compatible and at least as high in the partial ordering, which is assumed to exist, for that parameter type.

Lemma 1 can be applied immediately and the comparison defines a partial ordering. A (machine readable) example of node attributes from [2] for contours of simple shapes is: ***A[@id=ref;ext:sqr:cg(64,86)].***

The attribute string shows the graph identity is ***ref***, the contour is external, is judged to be a square, and has a centroid at ***(64,86)***. The terms ***ext***, ***sqr*** and ***cg(64,86)*** are propositions; the last one having the parameters ***64, 86***.

The partial ordering of parameters will be exploited in future work where the attribute strings will include the concept of a confidence. For example the centre of gravity attribute, currently ***cg(64,86)***, could be extended to include an estimated position and a confidence. A term such as ***cg(64±5, 86±7)*** would be judged to be compatible with, but less precise than ***cg(65±1, 87±1)***.

Attributed relational graphs can be decomposed into unattributed graphs and attribute strings. Lemma 1 establishes that attributed graphs can be partially ordered by the following: two attributed graphs $g_1=<N_1,A_1>$ and $g_2=<N_2,A_2>$ are partially ordered with $g_1 \subseteq_{Dg} g_2$ iff

1. the construction of two unattributed graphs, ug_1 and ug_2, where: $ug_1 = <$ ***First***(N_1), ***First***$(A_1)>$ and $ug_2 = <$ ***First***(N_2), ***First***$(A_2)>$ has $ug_1 \subseteq_{Ug} ug_2$. and
2. the attribute strings of corresponding nodes and arcs identified in **1**. above are similarly ordered.

Thus if n_1 and n_2 are corresponding nodes from g_1 and g_2 then ***Second***$(n_1) \subseteq_{At}$ ***Second***(n_2), and if a_1 and a_2 are corresponding arcs from g_1 and g_2 then ***Second***$(a_1) \subseteq_{At}$ ***Second***(a_2). Lemma 1 was not directly applied to N_1, N_2 and A_1, A_2 because it would restrict the freedom allowed in [6] to compose fragments of g_2 as part of the comparison with g_1. This accommodates missing detail.

2.2. THE MATCH_TABLE TYPE AND ITS PARTIAL ORDERING

The match_table data type contains elements that define an association between two graphs. By comparing two elements of the match_table type a comparison is being made between the strengths of the matches between the two graphs. Each element of match_table has two components: ***pn***, a list of nodepairs, and ***pa***, a list of arc pairs. Thus if ***m*** belongs to the match_table data type, and represents an association between

graphs g_x and g_y, we have $m = < pn, pa >$ where $pn = < \ldots , <(n_x)_i, (n_y)_j>, \ldots >$ with $(n_x)_i$ and $(n_y)_j$ being the i^{th} node from g_x and the j^{th} node from g_y respectively, and $pa = < \ldots , <(a_x)_i, (a_y)_j>, \ldots >$ with $(a_x)_i$ and $(a_y)_j$ being the i^{th} arc from g_x and the j^{th} arc from g_y respectively.

The definitions below are made more clear if the following convention is used for representing nodes. A node written as ${}_pn_{gL}$ is a node involved in the p^{th}. match table, belonging to graph g, and is the leading (first) node in the representation of an arc as a <node,node> sequence.

The node represented as ${}_pn_{gT}$ is the trailing (second) node of the the two elements, m_1 and m_2, that represent different matchings of graphs g_x and g_y, are expressed as: $m_1 = < pn_1, pa_1 >$ and $m_2 = < pn_2, pa_2 >$. Using Lemma 1, a partial ordering for elements of match_table will exist such that: $m_1 \subseteq_{Dm} m_2$ iff $pn_1 \subseteq_{Dm} pn_2$ and $pa_1 \subseteq_{Dm} pa_2$.

The $\subseteq$ between the $pn's$ and $pa's$ would be the subset relation if it were not for the attributes. In order to define the comparison it must be recalled that the $pn's$ are sets of node pairs and that each node is itself a pair *<n_name, attribute_of_n_name>*.

Using the selectors we can define $pn_1 \subseteq_{Dpn} pn_2$ iff

- First(First(pn_1)) $\subseteq$ First(First(pn_2)) and for each n_name $\in$ First(First(pn_1)) there is qn_1 = <n_name, attribute$_1$_of_n_name> $\in$ First(pn_1) and qn_2 = <n_name, attribute$_2$_of_n_name> $\in$ First(pn$_2$) with attribute$_1$_of_n_name $\subseteq_{At}$ attribute$_2$_of_n_name and
- First(Second(pn_1)) $\subseteq$ First(Second(pn_2)) and for each n_name $\in$ First(Second(pn_1)) there is qn_1 = <n_name, attribute$_1$_of_n_name> $\in$ Second(pn_1) and qn_2 = <n_name, attribute$_2$_of_n_name> $\in$ Second(pn_2) with attribute$_1$_of_n_name $\subseteq_{At}$ attribute$_2$_of_n_name

and $pa_1 \subseteq_{Dpa} pa_2$ iff $pn_1 \subseteq_{Dpn} pn_2$ and defining

- pa_1=**{...**<<<$_1n_{xL},_1n_{xT}$>,$att\_{}_1n_{xL}\_{}_1n_{xT}$>, <<$_1n_{yL},_1n_{yT}$>,$att\_{}_1n_{yL}\_{}_1n_{yT}$>>>**...}** and
- pa_2=**{...**<<<$_2n_{xL},_2n_{xT}$>,$att\_{}_2n_{xL}\_{}_2n_{xT}$>, <<$_2n_{yL},_2n_{yT}$>,$att\_{}_2n_{yL}\_{}_2n_{yT}$>>>**...}** then
- if <$_1n_{xL},_1n_{yL}$>, <$_1n_{xT},_1n_{yT}$> $\hat{I}$ pn_1 and <$_2n_{xL},_2n_{yL}$>, <$_2n_{xT},_2n_{yT}$> $\hat{I}$ pn_2 with <$_1n_{xL},_1n_{yL}$> $\subseteq_{Dpn}$ <$_2n_{xL},_2n_{yL}$> and <$_1n_{xT},_1n_{yT}$> $\subseteq_{Dpn}$ <$_2n_{xT},_2n_{yT}$> and $att\_{}_1n_{xL}\_{}_1n_{xT} \subseteq_{At} att\_{}_2n_{xL}\_{}_2n_{xT}$ and $att\_{}_1n_{yL}\_{}_1n_{yT} \subseteq_{At} att\_{}_2n_{yL}\_{}_2n_{yT}$

The relation defined by the comparison is a partial ordering because of the reflexivity, transitivity and anti-symmetry of subset and the $\subseteq$ between attribute strings. This complicated definition essentially dismantles the node pairs and arc pairs and considers the parts separately. Lemma 1 was not directly applicable because $\subseteq_{Dpn}$ and $\subseteq_{Dpa}$ had not been previously defined.

2.3. THE CONSTRAINT TYPE AND ITS ORDERING

Constraints are conditions that must be satisfied before a problem can be solved. These can guide the solution by only considering associations that agree with the constraints, [1]. It is also possible to define cost functions that have have an acceptable value only when the constraint is satisfied, [4]. Using the lattice theory, constraints seem to be well expressed as classes of relations that partition the problem into sub-problems, [2]. In this case constraints have a role in scheduling the matching activities. The classes of constraints reflect physical properties of a given problem, but the method of defining the partial ordering is problem independent.

The partial ordering of the constraints is defined in terms of the sets of propositions contained in the constraints. The first step is to note that a constraint is in fact an attribute string: *constraint = { … < proposition, <p_1, <p_2,...>>> …}*

The individual <proposition,<..parameters..>> can be extracted using the selectors, First, Second, … N^{th} , etc. At present, only the proposition part of the constraint is considered. This is obtained by using the First selector.

Let the set of all possible propositions that can be found in a problem be $\boldsymbol{U}$. Let $\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ be sets $\boldsymbol{C_1}$ = First($\boldsymbol{c_1}$) and $\boldsymbol{C_2}$ = First($\boldsymbol{c_2}$).The partial ordering for constraints is defined: with $\boldsymbol{c_1}$, $\boldsymbol{c_2}$ $\hat{\boldsymbol{I}}$ CONSTRAINT, $\boldsymbol{c_1} \subseteq_{Dc} \boldsymbol{c_2}$ iff $\boldsymbol{(U - C_1)}$ $\acute{\boldsymbol{I}}$ $\boldsymbol{(U - C_2)}$. It immediately follows that $\subseteq_{Dc}$ is a partial ordering because $\subseteq$ is a partial ordering.

The three data types, graphs, match_table and constraints have had partial orderings defined for them. The next step is to consider a matching iteration.

3. AN EXAMPLE OF A MATCHING ITERATION

The matching of two graphs is a search to find a consistent association between them at an acceptable cost, [1]. In this section the control strategy is of particular interest.

The iteration attempts matching in partitions of the problem at successively higher levels in the domain of constraints, [2]. The solution of each partition is sub-optimal because it does not have access to the entire context of the match. The input graphs are $\boldsymbol{g_x}$ and $\boldsymbol{g_y}$; $\boldsymbol{c}$ is the constraint currently being considered.

Some service procedures must first be defined. Part($\boldsymbol{g_x}$,$\boldsymbol{c}$) extracts the subgraph of $\boldsymbol{g_x}$ that contains arcs not superior to $\boldsymbol{c}$ in the partial ordering of constraints. Nextinferior($\boldsymbol{c}$) is a procedure that returns a list of constraints that are immediately inferior to the given constraint, $\boldsymbol{c}$. Terminate($\boldsymbol{m}$) returns TRUE if the iteration has reached a sufficient level to terminate. This procedure needs to share memory with a supervising process so that

movements of the working point can be monitored and a (near) fixed point can be identified.

Graph_isomorphism attempts to match the selected subgraphs of $\boldsymbol{g}_x$ and $\boldsymbol{g}_y$, using any effective method having the properties defined in section 1. When Graph_isomorphism succeeds, the increment to the match_table is LUB'd with the initial version of the match_table. The result is returned as the improved state of the match. As with LUB, Graph_isomorphism returns T if an inconsistency is detected.

The detection of an inconsistency implies that an inappropriate association was made earlier in the matching. The system's response is to selectively delete existing matches until a consistent state is again obtained, and then to proceed with the iteration. The deletions are information destroying, are not computable in the Scott sense, and are not part of the convergence: they effectively implement backtracking.

The progress to convergence can be understood by imagining an ensemble of matching attempts. Those that happen to trigger deletions will have their working points fall in the partial orderings. Those that do not trigger deletions will tend to rise in the partial ordering. If there is, in fact, only a limited number of match processors, then resources are switched to give attention to the most promising attempts, [5]. These actions are included under 'reschedule' in the procedure below:

```
match(m,gx,gy,c):match_table
Begin
Var m1: match_table; nonconsist : boolean;
m1:=m;
IF NOT Terminate(m) Then
    Begin
        For each element, cx, in Nextinferior(c) Do
        Begin
                m1: = LUB(m1,match(m1,gx,gy,cx));
                nonconsist := (m1 = T);
                IF nonconsist THEN reschedule;
        End;
        m1: = graph_isomorphism(m1, Part(gx,c),Part(gy,c));
                nonconsist := (m1 = T);
        IF nonconsist THEN reschedule;
    End;
    return(m1);
End;
```

The match procedure would be invoked by result: = match(Empty_match_table, g_reference,g_unknown,T). The input graphs, g_reference and g_unknown, are,

respectively a prediction from a model of what is expected and a representation obtained from an input image. The recursive calls within match search the constraint space. The first constraints used in matching attempts are those that are most restrictive.

4. THE CONTEXT OF THE MATCHING ITERATION WITHIN THE CONTROL STRATEGY

Even given the partitioning of the problem by constraints, the matching can still be caught in the 'NP' trap. In this case the result of expending effort in 'match' and 'graph_isomorphism' is a fruitless setting and clearing of trial associations of graph fragments. In terms of the theory, the efforts produce no strengthening of the working point in the lattice.

One benefit of the partial ordering is that the control strategy can observe the movement of the working point in the lattice and make scheduling decisions based upon what it observes. This implies that multiple processes will be used in implementing a practical system, [5]. Cost information is also important in allocating resources The iteration of section 3 does not explicitly pass costs upwards from the graph_isomorphism activity to the original call. A pilot system using exactly this strategy, [8], writes the cost of each fragmentary association into the match_table elements. Preliminary experimental results, [5], [8], suggest that allocation of resources is particularly important when more than one reference class is being considered.

5. CONCLUDING REMARKS

The monitoring of the convergence of a discrete iteration as part of its control strategy should be helpful in a large class of algorithms that are NP. Graph matching is one example. Without scheduling resources in response to performance the prospects for tractable solutions to NP problems seem limited.

The requirements of defining a partial ordering and of showing that the functions are order preserving gives strong guidance to the software engineering of a system. For computer vision, the approach supports a formal treatment of combining items of evidence and the use of constraints to partition a problem, [2].

REFERENCES

1. D.H.Ballard, C.M.Brown, *Computer Vision, Chapters 10, 11*, Prentice Hall, 1982.

2. R.E.Blake, “Graph matching with a sorted table: a suboptimal method that is

effective and allows problem partitioning". *Proceedings of ICARCV 90,* Singapore, p.956-960, 1990.

3. D.S.Scott, "Lectures on a mathematical theory of computation", in *Theoretical Foundations of Programming Methodology*, Reidel, p. 145-292

4. M.G.Thomason, "Structural methods in pattern analysis", *Pattern Recognition Theory and Applications,* NATO ASI series volume 30, Springer Verlag, p.307-321, 1987.

5. R.E.Blake, "Development of an incremental graph matching device", *Pattern Recognition Theory and Applications,* NATO ASI series volume 30, Springer Verlag, p357-366, 1987.

6. R.E.Blake, "A partial ordering for relational graphs applicable to varying levels of detail", *Pattern Recognition Letters* **9**, p.305-312, 1990.

7. L.G.Shapiro, R.M.Haralick, "Matching relational structures using discrete relaxation", *Syntactic and Structural Pattern Recognition Theory and Applications,* World Scientific, p.177-195, 1987.

8. R.E.Blake, "Recognising objects by matching attributed relational graphs", *Proceedings of The Seventh Scandinavian Conference on Image Analysis,* Aalborg, Denmark, August, p.588-596, 1991.